Introduction to Neural Networks 2

Dr. Mahmoud N Mahmoud
mnmahmoud@ncat.edu

North Carolina A & T State University

October 7, 2020

Talk Overview

1. Backpropagation in Neural Networks

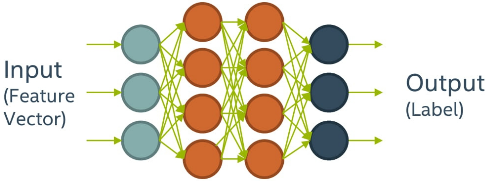
Outline

HOW TO TRAIN A NEURAL NET?

- Put in training inputs, get the output
- Compare output to correct answers: look at loss function J
- Adjust and repeat!
- Backpropagation tells us how to make a single adjustment using calculus.

HOW TO TRAIN A NEURAL NET?

- Put in training inputs, get the output
- Compare output to correct answers: look at loss function J
- Adjust and repeat!
- Backpropagation tells us how to make a single adjustment using calculus.



HOW HAVE WE TRAINED BEFORE?

Gradient Descent!

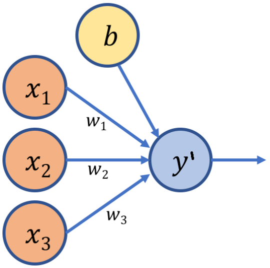
1. Make prediction
2. Calculate Loss
3. Calculate gradient of the loss function w.r.t. parameters
4. Update parameters by taking a step in the opposite direction
5. Iterate

HOW HAVE WE TRAINED BEFORE?

Gradient Descent!

1. **Make prediction**
2. **Calculate Loss**
3. Calculate gradient of the loss function w.r.t. parameters
4. Update parameters by taking a step in the opposite direction
5. Iterate

Logistic Regression as Neural Network (Refresher 1/5)

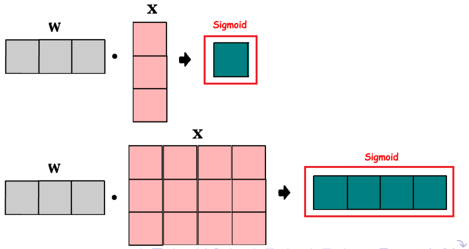


Feedforwad one sample

1. $z = \mathbf{w}^T \cdot \mathbf{x} + b$
2. $y = \sigma(z)$

Feedforwad batch

1. $\mathbf{z} = \mathbf{w} \cdot \mathbf{X} + b$
2. $\mathbf{y} = \sigma(\mathbf{z})$

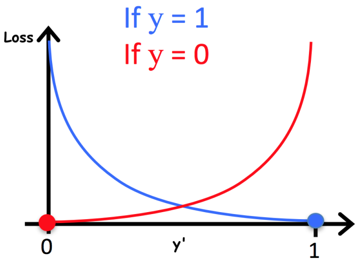


Logistic Regression as Neural Network (Refresher 2/5)

- Since y' in logistic regression is a probability between 0 and 1.

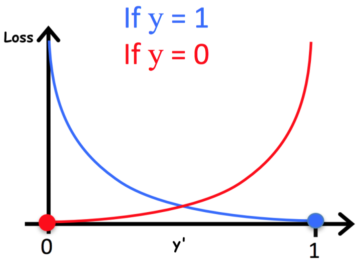
Logistic Regression as Neural Network (Refresher 2/5)

- Since y' in logistic regression is a probability between 0 and 1.
- Our loss can be defined with the following loss function.
  - if y = 1 : Loss = -log(y')
  - if y = 0 : Loss = -log(1-y')



Logistic Regression as Neural Network (Refresher 2/5)

- Since y' in logistic regression is a probability between 0 and 1.
- Our loss can be defined with the following loss function.
  - if $y = 1$: Loss = -log(y')
  - if $y = 0$: Loss = -log(1-y')



Loss$= \ell =$ -ylog(y')-(1-y)log(1-y')

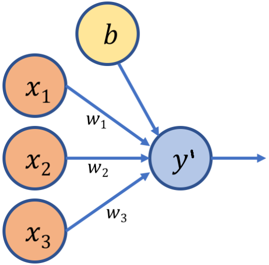
Logistic Regression as Neural Network (Refresher 3/5)

- For n features: $z = \sum_{i=0}^{i=n} w_i x_i$, (w_0 is the bias)
- vector representation $z = \mathbf{w}^T\mathbf{x}$
- $y = sigmoid(z) = \sigma(z)$
- $\ell = -ylog(y') - (1-y)log(1-y')$

Logistic Regression as Neural Network (Refresher 4/5)

$$
\begin{aligned}
\frac{d\ell}{dw_i} &= \frac{d\ell}{dy'}\frac{dy'}{dw_i} = \frac{d\ell}{dy'}\frac{dy'}{dz}\frac{dz}{dw_i} \\
&= \underbrace{\Big[\frac{-y}{\sigma(z)} + \frac{1-y}{1-\sigma(z)}\Big]}_{\frac{d\ell}{dy'}} * \underbrace{\sigma(z)(1-\sigma(z))}_{\frac{dy'}{dz}} * \underbrace{x_i}_{\frac{dz}{dw_i}} \\
&= \underbrace{\Big[\frac{-y(1-\sigma(z)) + (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\Big]}_{\frac{d\ell}{dy'}} * \underbrace{\sigma(z)(1-\sigma(z))}_{\frac{dy'}{dz}} * \underbrace{x_i}_{\frac{dz}{dw_i}} \\
&= \underbrace{\Big[\frac{-y(1-\sigma(z)) + (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\Big]}_{\frac{d\ell}{dy'}} * \underbrace{\sigma(z)(1-\sigma(z))}_{\frac{dy'}{dz}} * \underbrace{x_i}_{\frac{dz}{dw_i}} \\
&= (\sigma(z) - y) * x_i \\
&= (y' - y) * x_i
\end{aligned}
$$

Logistic Regression as Neural Network (Refresher 5/5)

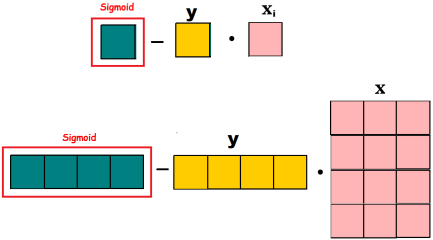


Loss one sample (using log loss)

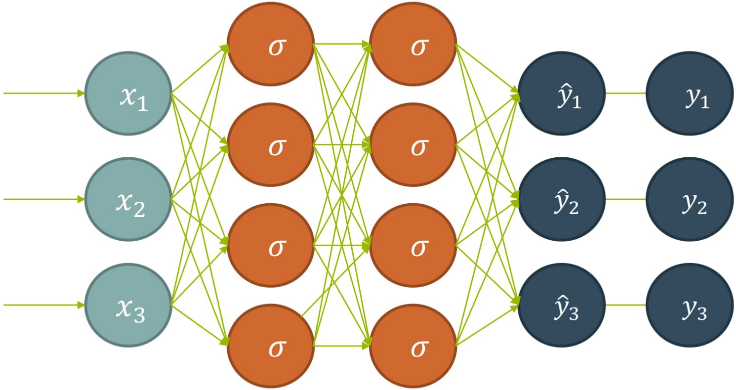
1. $\frac{d\ell}{dw_i} = (y' - y) * x_i$

Loss four samples (using log loss)

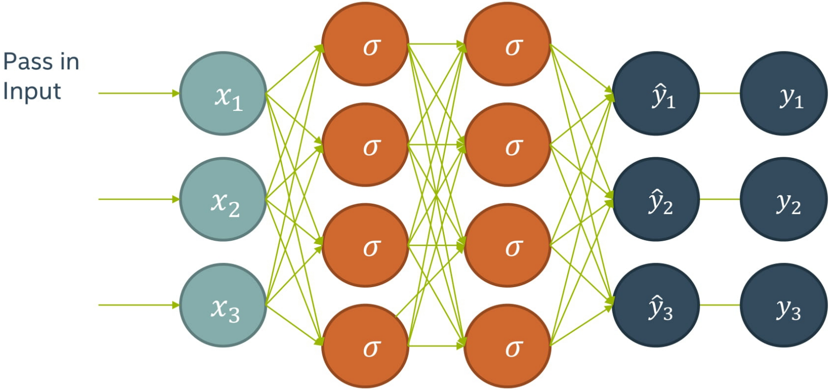
1. $\frac{d\ell}{d\mathbf{w}} = (\mathbf{y}' - \mathbf{y}) \cdot \mathbf{X}^T$



FEEDFORWARD NEURAL NETWORK

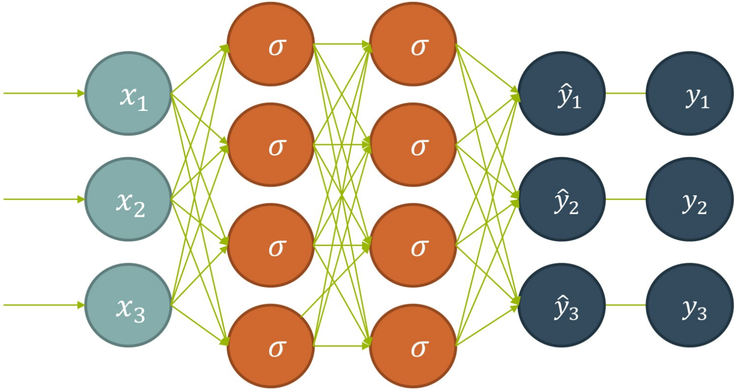


FORWARD PROPAGATION

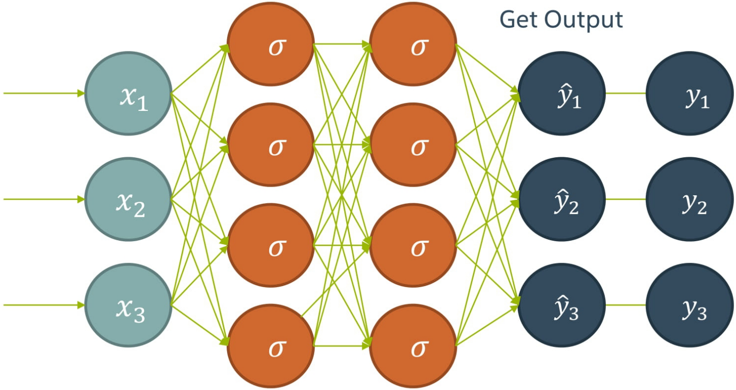


FORWARD PROPAGATION

Calculate each Layer

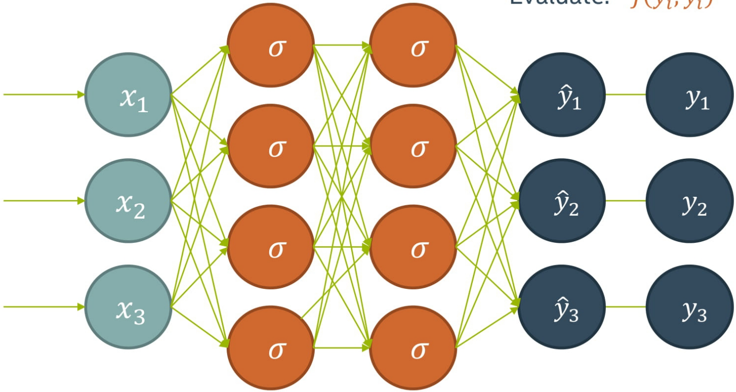


FORWARD PROPAGATION



FORWARD PROPAGATION

Evaluate: $J(y_i, \hat{y}_i)$



HOW HAVE WE TRAINED BEFORE?

Gradient Descent!

1. Make prediction
2. Calculate Loss
3. **Calculate gradient of the loss function w.r.t. parameters**
4. Update parameters by taking a step in the opposite direction
5. Iterate

HOW TO CALCULATE GRADIENT?

Chain rule

Chain Rule Refresher

- Foward propagation can be viewed as a long series of nested equations.
- **Backpropagation** is merely an application the Chain Rule to find the Derivatives of cost with respect to any variable in the nested equation. **Ex.**

  What is $\frac{df}{dx}$?

$$f(x) = A(B(C(x)))$$

Chain Rule Refresher

- Foward propagation can be viewed as a long series of nested equations.
- **Backpropagation** is merely an application the Chain Rule to find the Derivatives of cost with respect to any variable in the nested equation.
  Ex.

  What is $\frac{df}{dx}$?

$$f(x) = A(B(C(x)))$$

$$\frac{df}{dx} = \frac{dA}{dB} * \frac{dB}{dC} * \frac{dC}{dx}$$

HOW TO TRAIN A NEURAL NET?

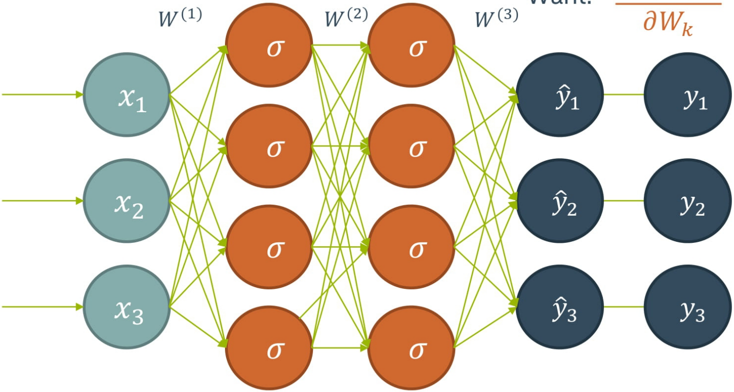
- How could we change the weights to make our Loss Function lower?
- Think of neural net as a function F: X -> Y
- F is a complex computation involving many weights W_k
- Given the structure, the weights “define” the function F (and therefore define our model)
- Loss Function is J(y,F(x))

HOW TO TRAIN A NEURAL NET?

- Get $\frac{\partial J}{\partial W_k}$ for every weight in the network.
- This tells us what direction to adjust each Wk if we want to lower our loss function.
- Make an adjustment and repeat!

FEEDFORWARD NEURAL NETWORK

Want: $\frac{\partial J(y_i, \hat{y}_i)}{\partial W_k}$



CALCULUS TO THE RESCUE

- Use calculus, chain rule, etc. etc.
- Functions are chosen to have "nice" derivatives
- Numerical issues to be considered

PUNCHLINE

$$\frac{\partial J}{\partial W^{(3)}} = (\hat{y} - y) \cdot a^{(3)}$$

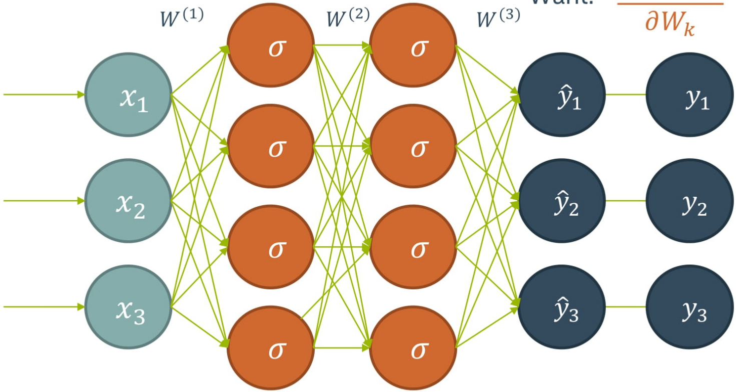
$$\frac{\partial J}{\partial W^{(2)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot a^{(2)}$$

$$\frac{\partial J}{\partial W^{(1)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot W^{(2)} \cdot \sigma'\left(z^{(2)}\right) \cdot X$$

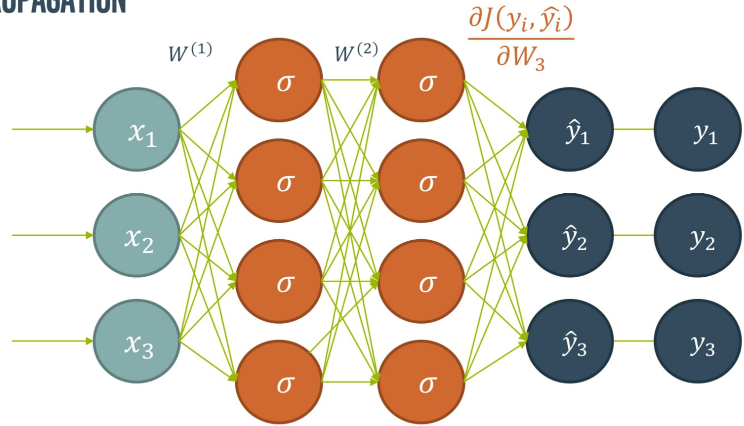
- Recall that: $\sigma'(z) = \sigma(z)(1-\sigma(z))$
- Though they appear complex, above are easy to compute!

BACKPROPAGATION

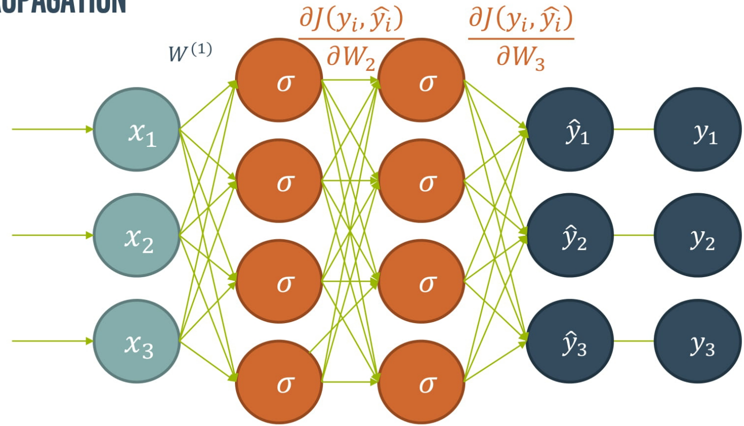
Want: $\dfrac{\partial J(y_i, \hat{y}_i)}{\partial W_k}$



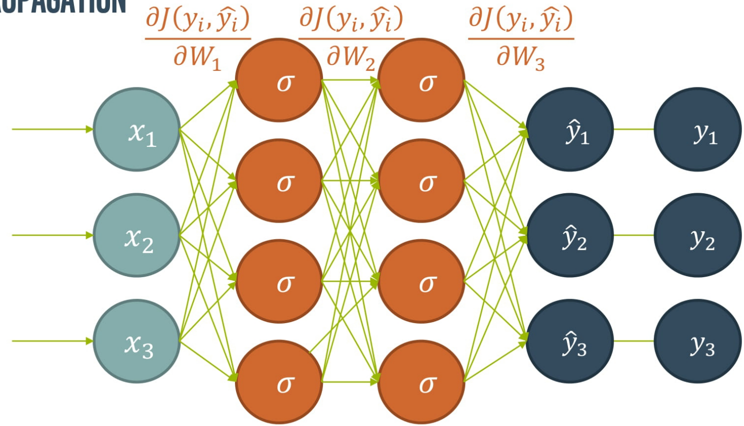
BACKPROPAGATION



BACKPROPAGATION



BACKPROPAGATION



HOW HAVE WE TRAINED BEFORE?

Gradient Descent!

1. Make prediction
2. Calculate Loss
3. Calculate gradient of the loss function w.r.t. parameters
4. **Update parameters by taking a step in the opposite direction**
5. Iterate

VANISHING GRADIENTS

Recall that:

$$\frac{\partial J}{\partial W^{(1)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot W^{(2)} \cdot \sigma'\left(z^{(2)}\right) \cdot X$$

- Remember: $\sigma'(z) = \sigma(z)(1-\sigma(z)) \leq .25$
- As we have more layers, the gradient gets very small at the early layers.
- This is known as the “vanishing gradient” problem.
- For this reason, other activations (such as ReLU) have become more common.

WHAT NEXT?

- **Given an example (or group of examples), we know how to compute the derivative for each weight.**
- **How exactly do we update the weights?**
- **How often? (after each training data point? after all the training data points?)**

WHAT NEXT?—GRADIENT DESCENT

- W_new = W_old - lr * derivative
- Classical approach—get derivative for entire data set, then take a step in that direction
- Pros: Each step is informed by all the data
- Cons: Very slow, especially as data gets big

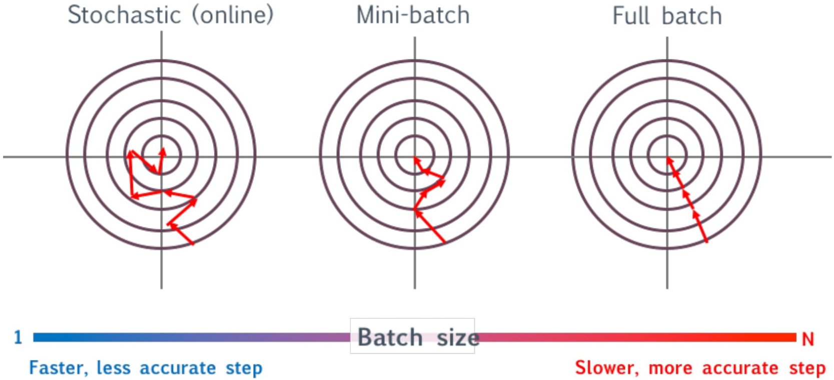
ANOTHER APPROACH: STOCHASTIC GRADIENT DESCENT

- Get derivative for just one point, and take a step in that direction
- Steps are “less informed” but you take more of them
- Should “balance out”
- Probably want a smaller step size
- Also helps “regularize”

COMPROMISE APPROACH: MINI-BATCH

- Get derivative for a "small" set of points, then take a step in that direction
- Typical mini batch sizes are 16, 32
- Strikes a balance between two extremes

COMPARISON OF BATCHING APPROACHES



BATCHING TERMINOLOGY

Full-batch:
Use entire data set to compute gradient before updating

Mini-batch:
Use a smaller portion of data (but more than single example) to compute gradient before updating

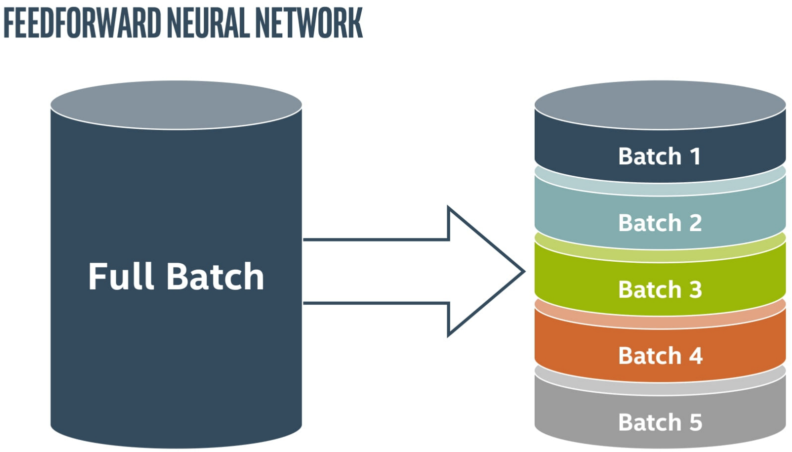
Stochastic Gradient Descent (SGD):
Use a single example to compute gradient before updating (though sometimes people use SGD to refer to minibatch, also)

BATCHING TERMINOLOGY

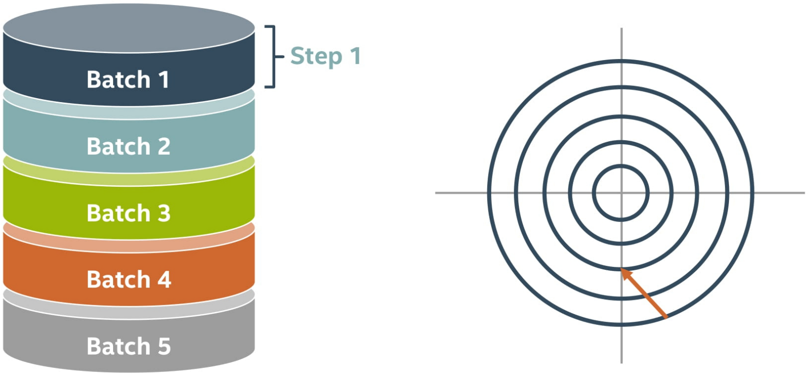
- An **Epoch** refers to a single pass through all of the training data.
- In full batch gradient descent, there would be one step taken per epoch.
- In SGD / Online learning, there would be n steps taken per epoch (n = training set size)
- In Minibatch there would be (n/batch size) steps taken per epoch
- When training, it is common to refer to the number of epochs needed for the model to be "trained".

NOTE ON DATA SHUFFLING

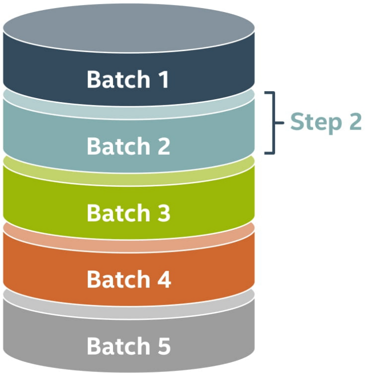
- **To avoid any cyclical movement and aid convergence, it is recommended to shuffle the data after each epoch.**
- **This way, the data is not seen in the same order every time, and the batches are not the exact same ones.**

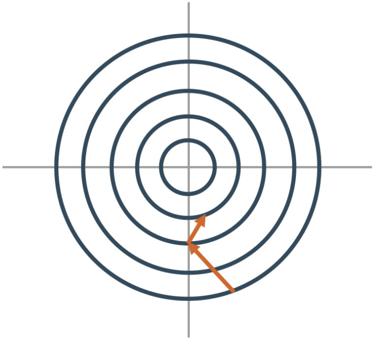

FEEDFORWARD NEURAL NETWORK
Full Batch
Batch 1
Batch 2
Batch 3
Batch 4
Batch 5

TRAINING IN ACTION

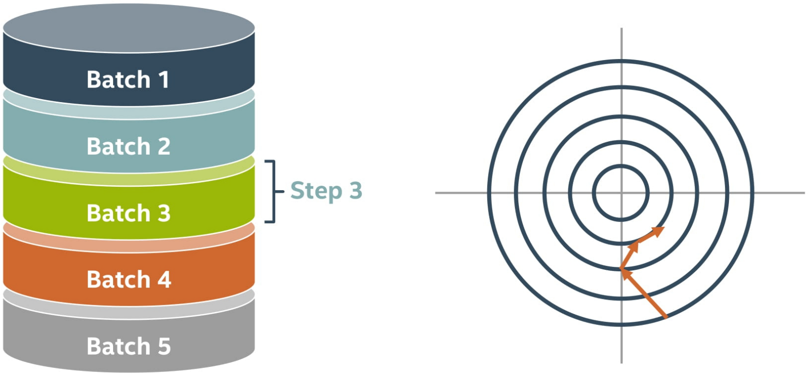


TRAINING IN ACTION

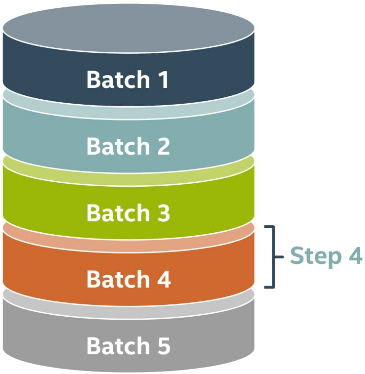


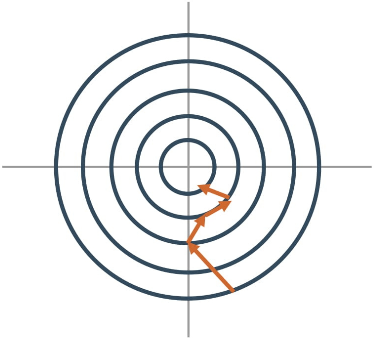


TRAINING IN ACTION

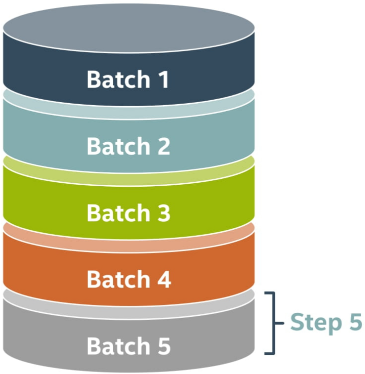


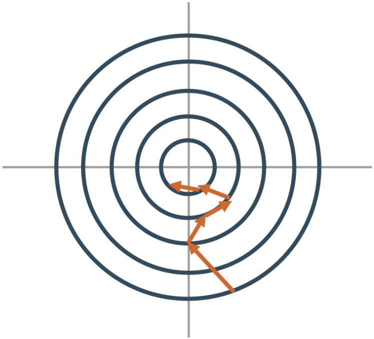
TRAINING IN ACTION

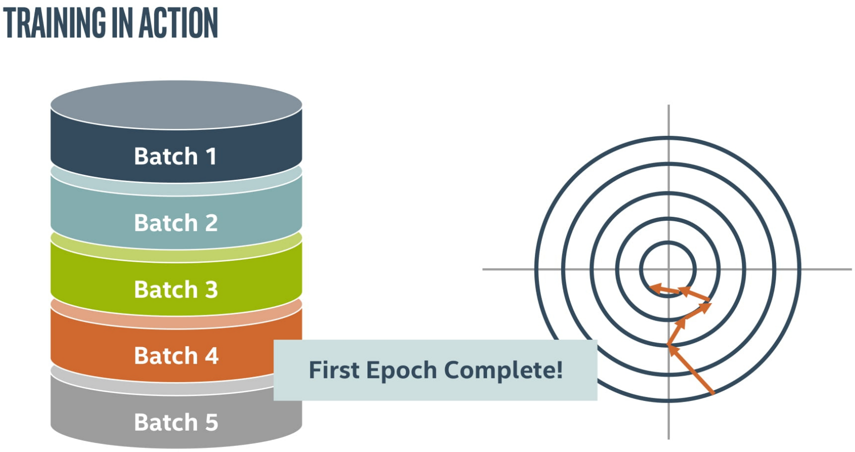


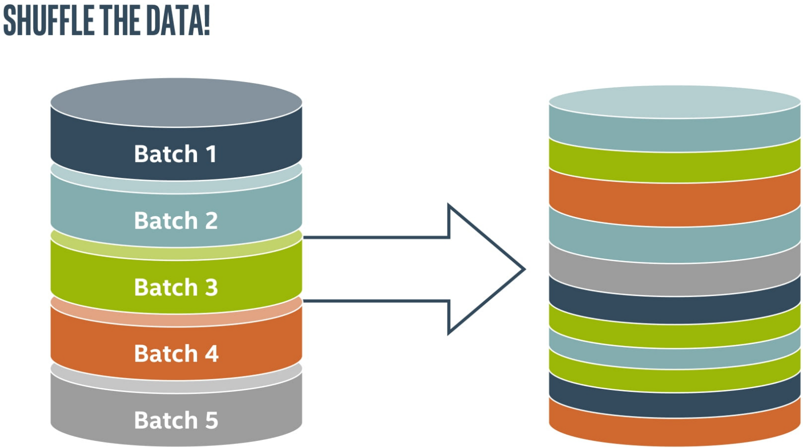


TRAINING IN ACTION

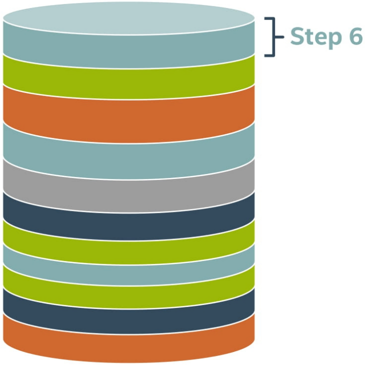


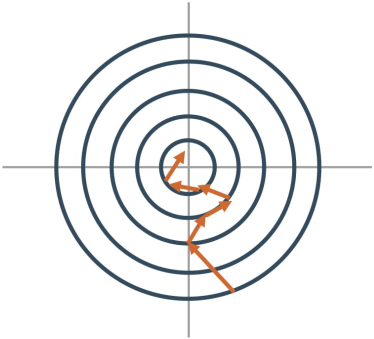



TRAINING IN ACTION
Batch 1
Batch 2
Batch 3
Batch 4
Batch 5
First Epoch Complete!


SHUFFLE THE DATA!
Batch 1
Batch 2
Batch 3
Batch 4
Batch 5

SHUFFLE THE DATA!





THE KERAS PACKAGE

- Keras allows easy construction, training, and execution of Deep Neural Networks
- Written in Python, and allows users to configure complicated models directly in Python
- Uses either Tensorflow or Theano “under the hood”
- Uses either CPU or GPU for computation
- Uses numpy data structures, and a similar command structure to scikit-learn (model.fit , model.predict, etc.)

TYPICAL COMMAND STRUCTURE IN KERAS

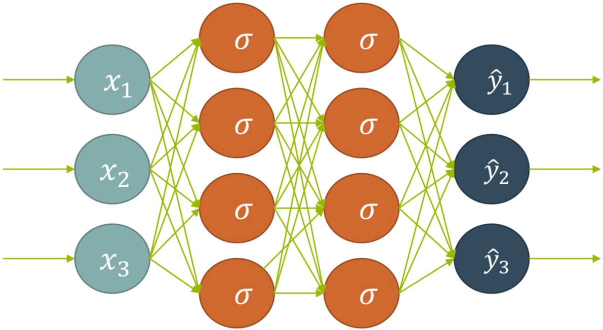
- **Build the structure of your network.**
- **Compile the model, specifying your loss function, metrics, and optimizer (which includes the learning rate).**
- **Fit the model on your training data (specifying batch size, number of epochs)**
- **Predict on new data**
- **Evaluate your results**

BUILDING THE MODEL

- Keras provides two approaches to building the structure of your model:
- Sequential Model: allows a linear stack of layers – simpler and more convenient if model has this form
- Functional API: more detailed and complex, but allows more complicated architectures
- We will focus on the Sequential Model.

RUNNING EXAMPLE, THIS TIME IN KERAS

Let's build this Neural Network structure shown below in Keras:



KERAS—SEQUENTIAL MODEL

First, import the Sequential function and initialize your model object:

```
from keras.models import Sequential
model = Sequential()
```

KERAS—SEQUENTIAL MODEL

Then we add layers to the model one by one.

```
from keras.layers import Dense, Activation

# For the first layer, specify the input dimension
model.add(Dense(units=4, input_dim=3))

# Specify an activation function
model.add(Activation(sigmoid'))

# For subsequent layers, the input dimension is presumed from
# the previous layer
model.add(Dense(units=4))
model.add(Activation(sigmoid'))
model.add(Dense(units=3))
model.add(Activation('softmax'))
```

MULTICLASS CLASSIFICATION WITH NEURAL NETWORKS

- **For binary classification problems, we have a final layer with a single node and a sigmoid activation.**
- **This has many desirable properties**
  - Gives an output strictly between 0 and 1
  - Can be interpreted as a probability
  - Derivative is “nice”
  - Analogous to logistic regression
- **Is there a natural extension of this to a multiclass setting?**

MULTICLASS CLASSIFICATION WITH NEURAL NETWORKS

- Reminder: one hot encoding for categories
- Take a vector with length equal to the number of categories
- Represent each category with one at a particular position (and zero everywhere else)

$$\begin{array}{ccc} \begin{bmatrix}1\\0\\0\end{bmatrix} & \begin{bmatrix}0\\1\\0\end{bmatrix} & \begin{bmatrix}0\\0\\1\end{bmatrix} \\ \textbf{Cat} & \textbf{Dog} & \textbf{Toaster} \end{array}$$

MULTICLASS CLASSIFICATION WITH NEURAL NETWORKS

- **For multiclass classification problems, let the final layer be a vector with length equal to the number of possible classes.**
- **Extension of sigmoid to multiclass is the softmax function.**
- $softmax(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$
- **Yields a vector with entries that are between 0 and 1, and sum to 1**

MULTICLASS CLASSIFICATION WITH NEURAL NETWORKS

- For loss function use "categorical cross entropy"
- This is just the log-loss function in disguise

$$C.E. = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

- Derivative has a nice property when used with softmax

$$\frac{\partial C.E.}{\partial softmax} \cdot \frac{\partial softmax}{\partial z_i} = \hat{y}_i - y_i$$

WAYS TO SCALE INPUTS

- **Linear scaling to the interval [0,1]**

$$x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

- **Linear scaling to the interval [-1,1]**

$$x_i = 2\left(\frac{x_i - \bar{x}}{x_{max} - x_{min}}\right) - 1$$

WAYS TO SCALE INPUTS

- **Standardization (making variable approx. std. normal)**

$$x_i = \frac{x_i - \bar{x}}{\sigma}; \qquad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Deep Learning Frameworks (1/2)

- TensorFlow

Deep Learning Frameworks (1/2)

- TensorFlow TensorFlow
  - Googles Tensorflow the most popular Deep Learning.

Deep Learning Frameworks (1/2)

- TensorFlow TensorFlow
    - Googles Tensorflow the most popular Deep Learning.
    - TF needs a lot of coding.

Deep Learning Frameworks (1/2)

- TensorFlow
  - Googles Tensorflow the most popular Deep Learning.
  - TF needs a lot of coding.
- PyTorch

Deep Learning Frameworks (1/2)

- TensorFlow
  - Googles Tensorflow the most popular Deep Learning.
  - TF needs a lot of coding.
- PyTorch
  - Was developed for Facebook services

Deep Learning Frameworks (1/2)

- TensorFlow
    - Googles Tensorflow the most popular Deep Learning.
    - TF needs a lot of coding.
- PyTorch
    - Was developed for Facebook services
    - In PyTorch, you can use standard debuggers, for example, pdb or PyCharm.

Deep Learning Frameworks (1/2)

- TensorFlow
  - Googles Tensorflow the most popular Deep Learning.
  - TF needs a lot of coding.
- PyTorch
  - Was developed for Facebook services
  - In PyTorch, you can use standard debuggers, for example, pdb or PyCharm.
- Sonnet

Deep Learning Frameworks (1/2)

- TensorFlow
  - Googles Tensorflow the most popular Deep Learning.
  - TF needs a lot of coding.
- PyTorch
  - Was developed for Facebook services
  - In PyTorch, you can use standard debuggers, for example, pdb or PyCharm.
- Sonnet
  - Built on top of TensorFlow.

Deep Learning Frameworks (1/2)

- TensorFlow TensorFlow
  - Googles Tensorflow the most popular Deep Learning.
  - TF needs a lot of coding.
- PyTorch PYTORCH
  - Was developed for Facebook services
  - In PyTorch, you can use standard debuggers, for example, pdb or PyCharm.
- Sonnet Sonnet
  - Built on top of TensorFlow.
  - High-level object-oriented libraries.

Deep Learning Frameworks (1/2)

- TensorFlow TensorFlow
  - Googles Tensorflow the most popular Deep Learning.
  - TF needs a lot of coding.
- PyTorch PYTORCH
  - Was developed for Facebook services
  - In PyTorch, you can use standard debuggers, for example, pdb or PyCharm.
- Sonnet Sonnet
  - Built on top of TensorFlow.
  - High-level object-oriented libraries.
- Keras K Keras

Deep Learning Frameworks (1/2)

- TensorFlow TensorFlow
  - Googles Tensorflow the most popular Deep Learning.
  - TF needs a lot of coding.
- PyTorch PYTORCH
  - Was developed for Facebook services
  - In PyTorch, you can use standard debuggers, for example, pdb or PyCharm.
- Sonnet Sonnet
  - Built on top of TensorFlow.
  - High-level object-oriented libraries.
- Keras K Keras
  - The best Deep Learning framework for those who are just starting out.

Deep Learning Frameworks (1/2)

- TensorFlow TensorFlow
  - Googles Tensorflow the most popular Deep Learning.
  - TF needs a lot of coding.
- PyTorch PYTORCH
  - Was developed for Facebook services
  - In PyTorch, you can use standard debuggers, for example, pdb or PyCharm.
- Sonnet Sonnet
  - Built on top of TensorFlow.
  - High-level object-oriented libraries.
- Keras K Keras
  - The best Deep Learning framework for those who are just starting out.
  - High level like Sonnet.

Deep Learning Frameworks (1/2)

- TensorFlow
  - Googles Tensorflow the most popular Deep Learning.
  - TF needs a lot of coding.
- PyTorch
  - Was developed for Facebook services
  - In PyTorch, you can use standard debuggers, for example, pdb or PyCharm.
- Sonnet
  - Built on top of TensorFlow.
  - High-level object-oriented libraries.
- Keras
  - The best Deep Learning framework for those who are just starting out.
  - High level like Sonnet.
- MXNet

Deep Learning Frameworks (1/2)

- TensorFlow TensorFlow
    - Googles Tensorflow the most popular Deep Learning.
    - TF needs a lot of coding.
- PyTorch PYTORCH
    - Was developed for Facebook services
    - In PyTorch, you can use standard debuggers, for example, pdb or PyCharm.
- Sonnet Sonnet
    - Built on top of TensorFlow.
    - High-level object-oriented libraries.
- Keras K Keras
    - The best Deep Learning framework for those who are just starting out.
    - High level like Sonnet.
- MXNet mxnet
    - Effectively parallel on multiple GPUs and many machines..

Deep Learning Frameworks (2/2)

- Gluon

Deep Learning Frameworks (2/2)

- Gluon GLUON
  - Gluon is based on MXNet and offers a simple API that simplifies the creation of deep learning models.

Deep Learning Frameworks (2/2)

- Gluon GLUON
    - Gluon is based on MXNet and offers a simple API that simplifies the creation of deep learning models.
- Swift SWIFT

Deep Learning Frameworks (2/2)

- Gluon GLUON
    - Gluon is based on MXNet and offers a simple API that simplifies the creation of deep learning models.
- Swift SWIFT
    - Swift for Tensorflow.

Deep Learning Frameworks (2/2)

- Gluon
  - Gluon is based on MXNet and offers a simple API that simplifies the creation of deep learning models.
- Swift
  - Swift for Tensorflow.
- Chainer

Deep Learning Frameworks (2/2)

- Gluon GLUON
  - Gluon is based on MXNet and offers a simple API that simplifies the creation of deep learning models.
- Swift SWIFT
  - Swift for Tensorflow.
- Chainer Chainer
  - The code is written in pure Python on top of the Numpy and CuPy libraries. (fast)

Deep Learning Frameworks (2/2)

- Gluon GLUON
  - Gluon is based on MXNet and offers a simple API that simplifies the creation of deep learning models.
- Swift SWIFT
  - Swift for Tensorflow.
- Chainer Chainer
  - The code is written in pure Python on top of the Numpy and CuPy libraries. (fast)
- DL4J DL4J

Deep Learning Frameworks (2/2)

- Gluon GLUON
  - Gluon is based on MXNet and offers a simple API that simplifies the creation of deep learning models.
- Swift SWIFT
  - Swift for Tensorflow.
- Chainer Chainer
  - The code is written in pure Python on top of the Numpy and CuPy libraries. (fast)
- DL4J DL4J
  - Deep Learning for Java.

Deep Learning Frameworks (2/2)

- Gluon
  - Gluon is based on MXNet and offers a simple API that simplifies the creation of deep learning models.
- Swift
  - Swift for Tensorflow.
- Chainer
  - The code is written in pure Python on top of the Numpy and CuPy libraries. (fast)
- DL4J
  - Deep Learning for Java.
- ONNX

Deep Learning Frameworks (2/2)

- Gluon GLUON
  - Gluon is based on MXNet and offers a simple API that simplifies the creation of deep learning models.
- Swift SWIFT
  - Swift for Tensorflow.
- Chainer Chainer
  - The code is written in pure Python on top of the Numpy and CuPy libraries. (fast)
- DL4J DL4J
  - Deep Learning for Java.
- ONNX ONNX
  - Enables models to be trained in one framework and transferred to another for inference.

References

Intel Nervana AI Academy
https://software.intel.com/content/www/us/en/develop/training

Questions